

# Tercer Parcialito

## Ejercicio 1

Construir un conjunto de segmentos y un mapa de paginación (un solo directorio y varias tablas de página), tal que las siguientes traducciones sean válidas:

| Lógica            | Lineal     | Física     | Acción                |
|-------------------|------------|------------|-----------------------|
| 0x0010:0x000391AB | 0x000D2B44 | 0x20000B44 | Leer 4KB de código    |
| 0x0010:0x94752B74 | 0x947EC50D | 0xC7A9D50D | Ejecutar 2B de código |
| 0x0030:0x000DDDDD | 0x000DDDDD | 0x20000DDD | Leer 8KB de datos     |
| 0x0040:0x0014FFFF | 0xAAAAAFFF | 0xE7ABBFFF | Escribir 4B de datos  |

Si no es posible completar alguna traducción, justificar, dejando clara cuál es la razón que la traducción incompatible. Por simplicidad considerar que todas las traducciones corresponden a acciones realizadas en nivel cero. Se pide además que el límite de los segmentos creados en segmentación sea el mínimo posible.

### Solución:

#### SEGMENTACIÓN

SELECTOR DE SEGMENTO:  $\begin{matrix} 15 & & 3 & 2 & 0 \\ | & \text{INDEX} & | & \text{TI} & | & \text{RPL} \end{matrix}$

- 0x0010 → 0 0000 0010 0 00 → INDICE=0x02 TI=0 RPL=00
- 0x0030 → 0 0000 0110 0 00 → INDICE=0x06 TI=0 RPL=00
- 0x0040 → 0 0000 1000 0 00 → INDICE=0x08 TI=0 RPL=00

#### SEGMENTO 0x2:

- BASE = LINEAL - OFFSET  
BASE = 0x000D2B44 - 0x000391AB = 0x00099999  
BASE = 0x947EC50D - 0x94752B74 = 0x00099999
- LIMIT = OFFSET + TAMAÑO DE LA ACCION - 1  
LIMIT = 0x000391AB + 0x1000 - 1 = 0x0003A1AA  
LIMIT = 0x94752B74 + 0x2 - 1 = 0x94752B75 → 0x94752FFF → LIMIT = 0x94752, G = 1  
Me quedo con este límite porque incluye al anterior

#### SEGMENTO 0x6:

- BASE = LINEAL - OFFSET  
BASE = 0x000DDDDD - 0x000DDDDD = 0x0
- LIMIT = OFFSET + TAMAÑO DE LA ACCION - 1  
LIMIT = 0x000DDDDD + 0x2000 - 1 = 0x000DFDDC

#### SEGMENTO 0x8:

- BASE = LINEAL - OFFSET  
BASE = 0xAAAAAFFF - 0x0014FFFF = 0xAA95B000
- LIMIT = OFFSET + TAMAÑO DE LA ACCION - 1  
LIMIT = 0x0014FFFF + 4 - 1 = 0x00150002 → 0x00150FFF → LIMIT = 0x00150, G = 1

| <i>Índice</i> | <i>Base</i> | <i>Límite</i> | <i>P</i> | <i>DPL</i> | <i>S</i> | <i>D/B</i> | <i>L</i> | <i>G</i> | <i>Tipo</i>              |
|---------------|-------------|---------------|----------|------------|----------|------------|----------|----------|--------------------------|
| 0x02          | 0x00099999  | 0x94752       | 1        | 0          | 1        | 1          | 0        | 1        | 0xA (Code, execute/read) |
| 0x06          | 0x00000000  | 0xDFDDC       | 1        | 0          | 1        | 1          | 0        | 0        | 0x0 (Data, read-only)    |
| 0x08          | 0xAA95B000  | 0x00150       | 1        | 0          | 1        | 1          | 0        | 1        | 0x2 (Data, read/write)   |

Cuadro 1: Descriptores de la GDT

## PAGINACIÓN

DIRECCION LINEAL:  $\overset{31}{\text{DIRECTORY}} \mid \overset{22}{\text{TABLE}} \mid \overset{12}{\text{OFFSET}} \mid \overset{0}{\text{}}$

- 0x000D2B44 = 000;0D2;B44
- 0x947EC50D = 251;3EC;50D
- 0x000DDDDD = 000;0DD;DDD
- 0xAAAAAFFF = 2AA;2AA;FFF

| <i>Rango Lineal</i>     | <i>Rango Físico</i>     | <i>R/W</i> | <i>U/S</i> | <i>P</i> |
|-------------------------|-------------------------|------------|------------|----------|
| 0x000D2000 a 0x000D2FFF | 0x20000000 a 0x20000FFF | 1          | 0          | 1        |
| 0x947EC000 a 0x947ECFFF | 0xC7A9D000 a 0xC7A9DFFF | 1          | 0          | 1        |
| 0x000DD000 a 0x000DDFFF | 0x20000000 a 0x20000FFF | 0          | 0          | 1        |
| 0xAAAAA000 a 0xAAAAAFFF | 0xE7ABB000 a 0xE7ABBFFF | 1          | 0          | 1        |

Cuadro 2: Esquema de paginación

| <i>PD</i> | <i>TABLA</i> | <i>RW</i> | <i>US</i> |
|-----------|--------------|-----------|-----------|
| 0x000     | PT1          | 1         | 0         |
| ...       | ...          | ...       | ...       |
| ...       | ...          | ...       | ...       |
| 0x251     | PT2          | 1         | 0         |
| ...       | ...          | ...       | ...       |
| 0x2AA     | PT3          | 1         | 0         |
| ...       | ...          | ...       | ...       |
| 0x3FF     | ...          | ...       | ...       |

| <i>PT1</i> |         | <i>RW</i> | <i>US</i> |
|------------|---------|-----------|-----------|
| 0x000      | ...     | ..        | ..        |
| ...        | ...     | ...       | ...       |
| 0x0D2      | 0x20000 | 1         | 0         |
| ...        | ...     | ...       | ...       |
| 0x0DD      | 0x20000 | 0         | 0         |
| 0x0DE      | 0x20001 | 0         | 0         |
| ...        | ...     | ...       | ...       |
| 0x3FF      | ...     | ...       | ...       |

| <i>PT3</i> |         | <i>RW</i> | <i>US</i> |
|------------|---------|-----------|-----------|
| 0x000      | ...     | ..        | ..        |
| ...        | ...     | ...       | ...       |
| 0x2AA      | 0xE7ABB | 1         | 0         |
| ...        | ...     | ...       | ...       |
| 0x3FF      | ...     | ...       | ...       |

| <i>PT2</i> |         | <i>RW</i> | <i>US</i> |
|------------|---------|-----------|-----------|
| 0x000      | ...     | ..        | ..        |
| ...        | ...     | ...       | ...       |
| 0x3EC      | 0xC7A9D | 1         | 0         |
| ...        | ...     | ...       | ...       |
| 0x3FF      | ...     | ...       | ...       |

## Ejercicio 2

Suponer que se tiene un sistema con segmentación flat y paginación activa. Implementar en C la función `void* hasAliasing(uint32_t* cr3, void* lineal1, void* lineal2)` que se encargará de detectar dentro del mapa de paginación pasado como parámetro, si existen dos direcciones lineales que se traduzcan a la misma dirección física. En el caso que esto ocurra, se deberá retornar la dirección física encontrada y las dos direcciones lineales, modificando `lineal1` y `lineal2`. De ocurrir múltiples veces, se deberá retornar la primer coincidencia que sea encontrada, contando desde la dirección lineal cero. Además, de existir más de dos traducciones que apunten a la misma dirección física, se deberán retornar las dos numéricamente más chicas.

*Ejemplo:*

| Rango Lineal            | Rango Físico            |
|-------------------------|-------------------------|
| 0x00010000 a 0x00010FFF | 0xA000A000 a 0xA000AFFF |
| 0x00011000 a 0x00011FFF | 0xA000B000 a 0xA000BFFF |
| 0x00012000 a 0x00012FFF | 0xA000C000 a 0xA000CFFF |
| 0x00013000 a 0x00013FFF | 0xA000C000 a 0xA000CFFF |
| 0x00014000 a 0x00014FFF | 0xA000B000 a 0xA000BFFF |
| 0x00015000 a 0x00015FFF | 0xA000B000 a 0xA000BFFF |

```
phaddr = hasAliasing(rcr3(), &lineal1, &lineal2)
```

El resultado de la función será:

phaddr → 0xA000B000

lineal1 → 0x00011000

lineal2 → 0x00014000

### Solución:

```
void* hasAliasing(uint32_t* cr3, void* lineal1, void* lineal2){
    // Recorro todas las direcciones virtuales posibles,
    // comparando una con todas en cada paso.
    for(int i = 0; i < 1024 * 1024; i++){
        for(int j = 0; j < 1024 * 1024; j++){
            if(i!=j){
                uint32_t vir1 = i << 12;
                uint32_t vir2 = j << 12;
                if(is_mapped(vir1, &cr3) && is_mapped(vir2, &cr3) &&
                    (vir2phy(vir1, &cr3) == vir2phy(vir2, &cr3))){
                    void* dir_fisica = (void*) vir2phy(vir1);
                    lineal1 = (void*) vir1;
                    lineal2 = (void*) vir2;
                    return dir_fisica;
                }
            }
        }
    }
    return 0;
}
```

```
//Función auxiliar que indica si una dirección virtual está mapeada
bool is_mapped(uint32_t dir_virtual, uint32_t cr3){
    // Virtual = /directory/table/offset/
    //           /31-----22/21-12/11---0/
    uint32_t directory_index = dir_virtual >> 22;
    uint32_t table_index = (dir_virtual >> 12) & 0x000003FF;
    pd_entry* page_directory = (pd_entry*) cr3;

    if (page_directory[directory_index].present){
        pt_entry* page_table = (pt_entry*) (page_directory[directory_index].page_table_base << 12);
        if (page_table[table_index].present == 1){
            return true;
        }
    }else{
        return false;
    }
}

//Función auxiliar que devuelve la dirección física correspondiente
//a la dirección virtual pasada por parámetro.
//Pre: La dirección virtual está mapeada
uint32_t vir2phy(uint32_t dir_virtual, uint32_t cr3){
    uint32_t directory_index = dir_virtual >> 22;
    uint32_t table_index = (dir_virtual >> 12) & 0x000003FF;

    pd_entry* page_directory = (pd_entry*) cr3;

    pt_entry* page_table = (pt_entry*) (page_directory[directory_index].page_table_base << 12);

    return page_table[table_index].physical_address_base << 12 + (dir_virtual & 0xFFF);
}

//Structs que definen los atributos de las entradas del directorio y
//de las tablas de páginas PD_ENTRY y PT_ENTRY:
typedef struct str_page_directory_entry {
    uint8_t present:1;
    uint8_t read_write:1;           // 1 = read/write, 0 = read only
    uint8_t user_supervisor:1;      // 1 = user, 0 = supervisor
    uint8_t page_write_through:1;
    uint8_t page_cache_disabled:1;
    uint8_t accesed:1;
    uint8_t x:1;
    uint8_t page_size:1;            // 0 = 4Kb, 1 = 4Mb
    uint8_t ignored:1;
    uint8_t available:3;
    uint32_t page_table_base:20;
} __attribute__((__packed__)) pd_entry;
```

```
typedef struct str_page_table_entry {
    uint8_t present:1;
    uint8_t read_write:1;           // 1 = read/write, 0 = read only
    uint8_t user_supervisor:1;      // 1 = user, 0 = supervisor
    uint8_t page_write_through:1;
    uint8_t page_cache_disabled:1;
    uint8_t accessed:1;
    uint8_t dirty:1;
    uint8_t x:1;
    uint8_t global:1;
    uint8_t available:3;
    uint32_t physical_address_base:20;
} __attribute__((__packed__)) pt_entry;
```

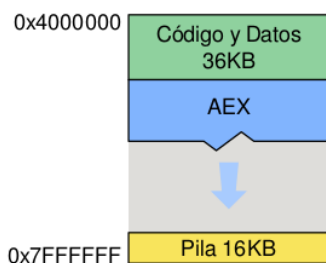
## Cuarto Parcialito

### Ejercicio 1

Suponer un sistema en modo protegido con segmentación *flat* y paginación activa. El sistema puede ejecutar hasta 10 tareas de usuario concurrentemente. Las tareas pueden estar activas o inactivas, las tareas que estén activas se ejecutaran todo el tiempo una después de la otra. Inicialmente todas las tareas estarán inactivas, y en ese caso se ejecutará una tarea especial denominada *Idle*. Para activar o desactivar tareas el usuario utiliza el teclado: cuando presiona los números del 0 al 9 puede activar o desactivar cada una de las tareas. Es decir, que estas comiencen a ejecutar o se detengan. A medida que se activen tareas utilizando el teclado, el sistema las ejecutará una a una en algún orden.

Como restricción del sistema, cuando una tarea es activada o desactivada, está debe comenzar a ser ejecutada o detenerse **inmediatamente**. No se debe esperar a que le toque su turno para ser ejecutada, o de estar corriendo, ser detenida e intercambiada por otra tarea.

El código y datos de cada tarea ocupa 36KB, mientras que el espacio requerido para la pila es de 16KB como mínimo. En direcciones virtuales cada tarea respeta el siguiente mapa:



El rango de memoria virtual entre que termina el código de la tarea y comienza la pila se denomina "área de expansión" (AEX).

El AEX corresponde a un área de memoria virtual que inicialmente no está mapeada a ninguna página de memoria física. A medida que las tareas soliciten memoria al sistema, el mismo se encargará de mapear paginas libres en la memoria física.

El sistema provee dos funcionalidades:

- `increaseExpansionArea`: Permite a cualquier tarea aumentar el tamaño del área de expansión mapeada, es decir, pedir memoria al sistema. Toma como parámetro la cantidad de bytes en que desea expandir el área, y retorna la cantidad de bytes en cuanto realmente se expandió el área.

Ejemplo: Se llama al servicio con el parámetro 2100, el resultado es 4096, ya que como mínimo se puede mapear una página.

- `getExpansionAreaLimit`: Este servicio permite a la tarea conocer cual es el límite de memoria del AEX a la que tiene acceso. Retorna la dirección del último byte al que puede acceder la tarea que llamo al servicio.

Ejemplo: Se llama al servicio para una tarea que aun no solicito ningún byte al sistema, el resultado será 4008FFF. Si luego la tarea pide 8KB de memoria y llama nuevamente al servicio, el resultado será 400AFFF.

Suponer que la memoria física total del sistema es de 128MB, de los cuales 110MB serán utilizados como un área libre. El área libre será utilizada para alojar las páginas de memoria que soliciten las tareas mediante el servicio `increaseExpansionArea`. Estas páginas serán utilizadas en algún orden y jamás podrá ser reutilizada una vez asignada a una tarea.

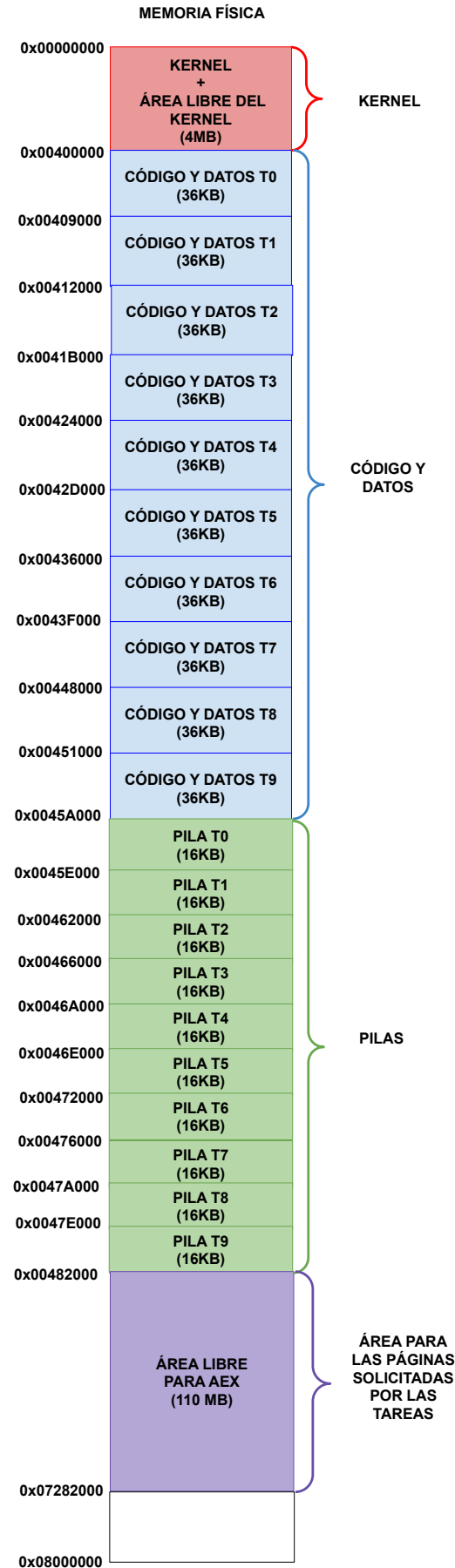
Se requiere:

1. Describir cómo se ubicaría en memoria física, las tareas, el kernel y el área libre. Enumerar para ello todos los rangos de memoria física que ocuparía cada parte de cada tarea, y del sistema. Elegir dos tareas cualesquiera y mostrar se como se realiza el mapeo sobre direcciones virtuales. Detallar cómo los rangos de memoria física mapean a direcciones virtuales mediante un esquema.
2. Implementar en ASM/C las rutinas de atención de interrupciones del reloj y teclado. Explicar detalladamente el mecanismo de activación y desactivación de tareas.
3. Diseñar y explicar el mecanismo de incrementado de memoria para tareas. Indicar cómo completaría las entradas en la IDT para los servicios `increaseExpansionArea` y `getExpansionAreaLimit`. Implementar en ASM/C ambas rutinas de atención de servicios del sistema.

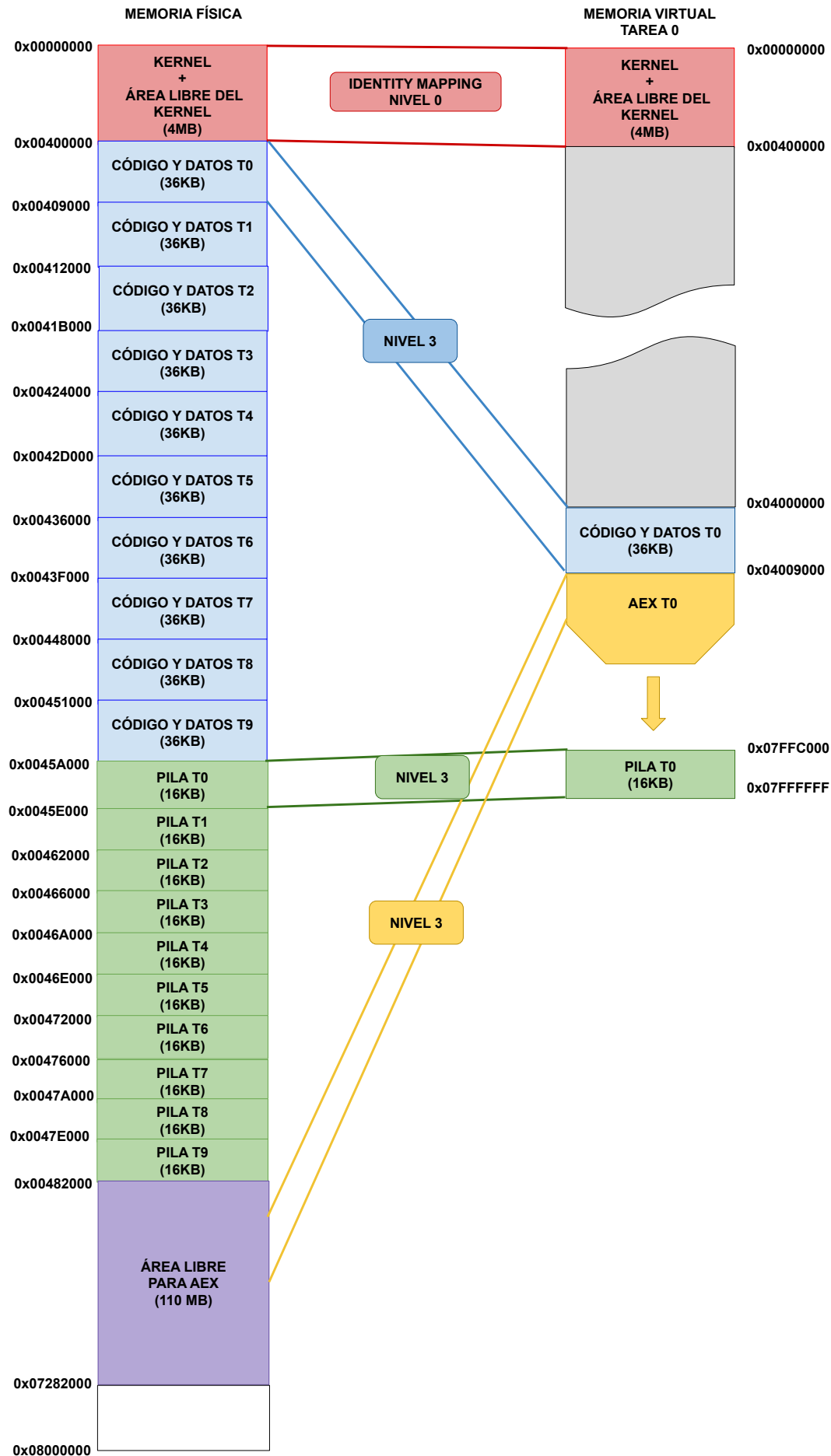
**Nota:** Detallar cualquier dato, estructura o variable referente al sistema que sea necesaria para lograr la implementación propuesta. Además implementar cualquier función auxiliar que requiera. Por ejemplo: La posición de los descriptores de TSS dentro de la GDT, o la implementación de una función para mapear páginas. En caso de requerir algún valor que no se encuentre definido de forma explícita por el enunciado, proponer un valor que considere razonable.

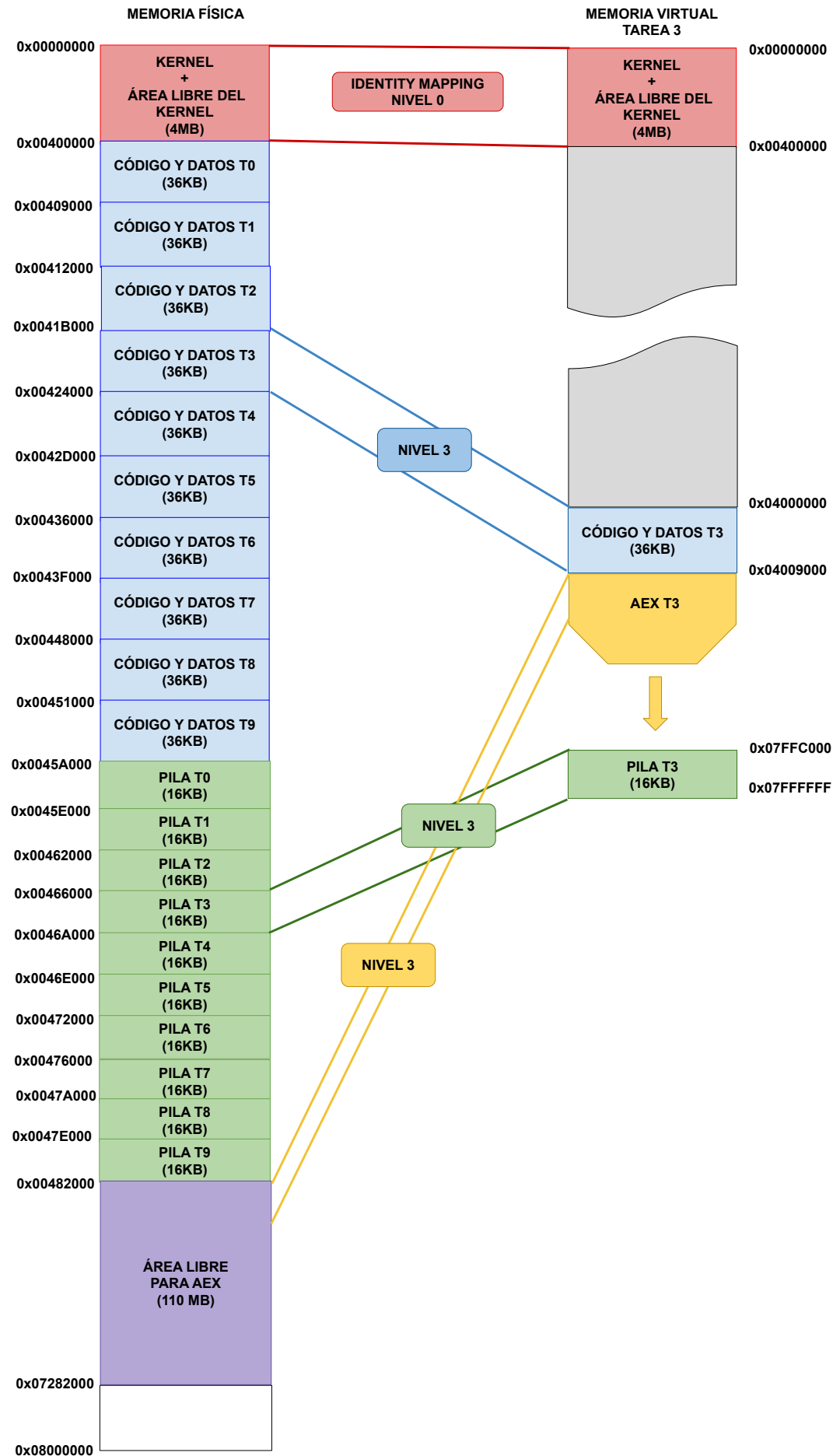
Solución:

1)









Nota: En los gráficos de la memoria virtual de la tarea 0 y tarea 3, el área virtual pintada de amarillo corresponde a qué mapeo realizaría si ya fue llamada la syscall `increaseExpansionArea`. Se quiso indicar aunque en el dibujo no quedó tan explícito, que cada tarea apunta a un área en memoria física en los 110MB designados diferente.

2)

Dado que se usa el esquema de segmentación flat, harán falta 4 descriptores en la GDT que referencien a segmentos que abarquen toda la memoria a usar, de código y datos de nivel 0 y 3 (usuario y kernel). También tengo que disponer en la GDT el descriptor de la tarea inicial, la tarea Idle, y de las 10 tareas. Debo también tener sus respectivas TSS donde almacenar el contexto del procesador al cambiar de tarea. Para manejar las interrupciones necesito una IDT que será un arreglo de descriptores. Ahí se va a definir un interrupt gate descriptor para cada interrupción. Las page directory y page table de cada tarea estarán dentro del área del kernel, así como las pilas de nivel 0.

- Asumo que cuento con la función `pic_finish1` que informa al PIC que ya se atendió la interrupción.
- Asumo que el descriptor TSS de la tarea Idle está en el índice 14 de la GDT.
- Asumo que los descriptores de las TSS de las 10 tareas están en las posiciones desde el índice 15 en adelante y que sus selectores están guardados en el atributo del struct `task_info` de cada tarea.
- Asumo que todas las tareas están cargadas e inactivas y que la tarea actual se inicializa en 0.
- Asumo que en la posición `GDT_CODE_RING_0` se encuentra el selector de segmento de código de nivel 0.
- Asumo que cuento con el struct `tss`

## Código ASM

```
extern activar_desactivar_tarea
extern pic_finish1
extern nextTask

offset: dd 0
selector: dw 0
; Rutina de reloj
global _isr33
_isr32:
    pushad
    call pic_finish1
    call nextTask
    ;Chequeo que la tarea actual no sea la misma que la siguiente
    str cx
    cmp ax, cx
    je .fin
    mov [selector], ax
    jmp far [offset]
.fin:
    popad
    iret

; Rutina de atención del teclado
global _isr33
_isr33:
    pushad
```

```
    in al, 0x60
    push eax
    call activar_desactivar_tarea
    add esp, 4
    call pic_finish1
    str cx
    cmp ax, cx
    je .fin
    mov [selector], ax
    jmp far [offset]
.fin:
    popad
    iret
```

## Código C

```
//Estructura que indica si la tarea está activada o desactivada, y su selector.
typedef struct str_task_info {
    uint8_t activa;
    int16_t selector_tss;
} __attribute__((__packed__)) task_info;
//Las tareas están cargadas en orden T0...T9 en cada uno de los arreglos.
#define CANT_TAREAS 10
tss TSSs[CANT_TAREAS];
task_info tareas[CANT_TAREAS];
int tarea_actual;

int16_t nextTask() {
    uint8_t i = tarea_actual+1;
    // Busco la siguiente tarea activa
    while (i != tarea_actual && !tareas[i].activa)
    {
        i = (i + 1) % CANT_TAREAS;
    }
    // Chequeo que haya una tarea activa, sino salto a la idle.
    if (tareas[i].esta_activa){
        // Actualizo la tarea actual y devuelvo su selector de tss
        tarea_actual = i;
        res = tareas[i].selector_tss;
    }else{
        res = 0x70; // tarea idle 14 << 3;
    }
    return res;
}

//El scanCode de las teclas 1 a 9 corresponden a su valor +1,
//en el caso de la tecla 0 su scanCode es 11.
int16_t activar_desactivar_tarea(int8_t scanCode){
    if(!(scanCode & 0x80)){ //Si se presionó
        if((1 < scanCode) && (scanCode < 12)){ //Si fue una tecla del 0 al 9.
            if(scanCode == 11){ //Si se presionó la tecla 0
                if(tareas[0].activa){
                    tareas[0].activa = 0; //Si estaba activa la desactivo
                    //Devuelvo el selector de la siguiente tarea a ser ejecutada
```

```
        return nextTask();
    }else{
        tareas[0].activa = 1; //Sino la activo
        //Devuelvo su selector para saltar a la tarea
        tarea_actual = 0;
        return tareas[0].selector_tss;
    }
}
}else{
    if(tareas[scanCode-1].activa){
        tareas[scanCode-1].activa = 0; //Si estaba activa la desactivo
        return nextTask();
    }else{
        tareas[scanCode-1].activa = 1;
        tarea_actual = scanCode-1;
        return tareas[scanCode-1].selector_tss;
    }
}
}
}
}
```

Entradas en la IDT:

- 0x20 (interrupción de reloj):
  - OFFSET: `&_isr32`
  - SELECTOR: `GDT_CODE_RING_0 << 3`
  - ATTR: `0x8E00` (DPL=0, P=1, D=1 (32bits))
- 0x21 (interrupción de teclado):
  - OFFSET: `&_isr33`
  - SELECTOR: `GDT_CODE_RING_0 << 3`
  - ATTR: `0x8E00` (DPL=0, P=1, D=1 (32bits))

### 3)

La idea es tener una variable donde almacenar la siguiente página física libre dentro del área de los 110 MB donde las tareas van a solicitar memoria, y para cada tarea su siguiente página libre en el rango virtual entre donde termina el código y datos de la tarea y su pila. Cada vez que la tarea solicite memoria llama a la función `pedirBytes`, que mapea la cantidad de páginas que solicite la tarea. Si la tarea pide más memoria de la que hay disponible, mapeo la mayor cantidad de páginas posibles.

Luego para conocer cuál es el límite de memoria del AEX de la tarea, solo resto 1 al valor de su siguiente página libre en el rango virtual de AEX.

- Asumo que el valor inicial de `sig_fisica_libre_aex` es `0x00482000` y el valor inicial para cada tarea de `sig_virtual_libre_aex` es `0x00409000`.
- Asumo que la interrupción `increaseExpansionArea` pasa el parámetro de la cantidad de bytes que solicita la tarea por el registro `eax`, y que ambas syscalls retornan sus resultados a través del registro `eax`.
- Asumo una función `mmu_nextFreeKernelPage()` que me devuelve una página libre dentro del kernel donde poder guardar una nueva página table.
- Asumo la existencia de una rutina `uint32_t rcr3(void)` que lee el valor contenido en el `cr3`.

### Código ASM

```
extern pedirBytes
extern limite_aex
syscall_ret: dd 0

sys_increaseExpansionArea:
    pushad
    push eax ;cantBytes
    call pedirBytes
    mov [syscall_ret], eax
    add esp, 4
    popad
    mov eax, [syscall_ret]
    iret

sys_getExpansionAreaLimit:
    pushad
    call limite_aex
    mov [syscall_ret], eax
    popad
    mov eax, [syscall_ret]
    iret
```

### Código C

```
#define USER 1
#define READWRITE 1
#define PRESENT 1
#define dir_tope_aex_tarea 0x07FFC000
#define dir_tope_aex 0x07282000

uint32_t sig_fisica_libre_aex;
//Arreglo donde en cada posición tengo la siguiente posición virtual libre de la tarea i.
uint32_t sig_virtual_libre_aex[CANT_TAREAS];

int pedirBytes(int cant_bytes){
    int cant_pags_pedidas;
    if(cant_bytes % 4096 != 0){
        //Redondeo a la siguiente página
        cant_pags_pedidas = (cant_bytes/4096) + 1;
    }else{
        cant_pags_pedidas = cant_bytes/4096;
    }
    //Si hay espacio disponible en la memoria física y en la virtual
    if(sig_fisica_libre_aex < dir_tope_aex &&
    sig_virtual_libre_aex[tarea_actual] < dir_tope_aex_tarea){
        int cant_pags_a_mapear;
        int pags_fisicas_disponibles = (dir_tope_aex - sig_fisica_libre_aex)/4096;
        //Si la tarea pide más páginas de las que puedo mapear, mapeo las posibles.
        if(pags_fisicas_disponibles < cant_pags_pedidas){
            cant_pags_a_mapear = pags_fisicas_disponibles;
        }else{
            cant_pags_a_mapear = cant_pags_pedidas;
        }
    }
```

```
    }
    //Mapeo la cantidad de páginas posibles segun cant_pags_a_mapear
    for(int i = 0; i < cant_pags_a_mapear; i++){
        mmu_mapPage(rcr3(), dame_virtual_libre(tarea_actual), dame_fisica_libre(), USER, RW);
    }
    return cant_pags_a_mapear * 4096;
}else{
    return 0;
}
}

uint32_t limit_aex(){
    return sig_virtual_libre_aex[tarea_actual]-1;
}

//Las demás entradas no mencionadas de las estructuras se dejan en 0.
void mmu_mapPage(uint32_t cr3, uint32_t virtual, uint32_t phy, uint32_t attr_us, uint32_t attr_rw){
    uint32_t directory_index = virtual >> 22;
    uint32_t table_index = (virtual >> 12) & 0x000003FF;

    pd_entry* page_directory = (pd_entry*) cr3;

    //Si la tabla no estaba presente la creo
    if (!page_directory[directory_index].present) { // Si present = 0
        uint32_t* nueva_pagina = (uint32_t*) mmu_nextFreeKernelPage();
        for (int i = 0; i < 1024; ++i) nueva_pagina[i] = 0;
        page_directory[directory_index].present = PRESENT;
        page_directory[directory_index].read_write = READWRITE; // Permiso menos restrictivo
        page_directory[directory_index].user_supervisor = attr_us;
        page_directory[directory_index].page_table_base = (uint32_t) nueva_pagina >> 12;
    }

    pt_entry* page_table = (pt_entry*) (page_directory[directory_index].page_table_base << 12);
    page_table[table_index].present = PRESENT;
    page_table[table_index].read_write = attr_rw;
    page_table[table_index].user_supervisor = attr_us;
    page_table[table_index].physical_address_base = phy >> 12;

    tlbflush(); //Invalida todas las entradas de la TLB (caché de traducciones)
}

uint32_t dame_fisica_libre(){
    uint32_t free_page = sig_fisica_libre_aex;
    sig_fisica_libre_aex += 0x1000;
    return free_page;
}

uint32_t dame_virtual_libre(int i){
    uint32_t free_page = sig_virtual_libre_aex[i];
    sig_virtual_libre_aex[i] += 0x1000;
    return free_page;
}
```

Entradas en la IDT de los servicios `increaseExpansionArea` y `getExpansionAreaLimit`:

- `increaseExpansionArea`:
  - OFFSET: `&sys_increaseExpansionArea`
  - SELECTOR: `GDT_CODE_RING_0 << 3`
  - ATTR: `0xEE00` (DPL=3, P=1, D=1 (32bits))
- `getExpansionAreaLimit`:
  - OFFSET: `&sys_getExpansionAreaLimit`
  - SELECTOR: `GDT_CODE_RING_0 << 3`
  - ATTR: `0xEE00` (DPL=3, P=1, D=1 (32bits))

Nota: Es importante que el DPL de estas interrupciones sea 3 (nivel usuario) ya que sino las tareas no podrían realizar las syscall. Como índices de las interrupciones en IDT aunque no lo especifico podrían ser cualquiera entre 32 y 255.



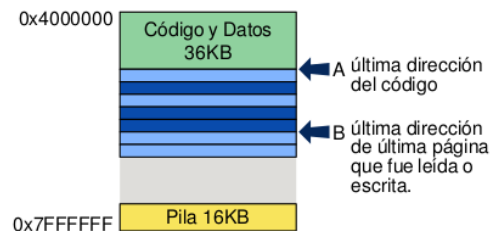
## Ejercicio 2

Suponer el sistema anterior.

Las tareas pueden solicitar toda la memoria que gusten, de esta forma, expanden su espacio direccionable. Puede darse el caso, que una tarea pida memoria pero jamas escriba ni lea en esta. Para detectar esta situación, cualquier tarea puede consultar la memoria realmente utilizada por cualquier otra tarea. Se denomina memoria utilizada a la memoria que alguna vez fue leída o escrita.

Se pide diseñar un servicio para el sistema que permita a una tarea consultar cuanta memoria fue utilizada por cualquier otra tarea. Para esto, el servicio deberá identificar la última pagina del área accesible de la tarea que fue leída o escrita, y retornar la cantidad de bytes entre el final del código de la tarea y la página encontrada.

Para el caso de la ilustración, los sectores coloreados en azul oscuro indican páginas leídas o escritas. El servicio retornaría la cantidad de bytes entre A y B.



### Solución:

- Asumo que para consultar sobre otra tarea, la tarea que llama a la syscall pasa como parámetro el índice de la tarea sobre la que quiere consultar en el registro `eax`.
- Para acceder a la dirección base de cada directorio de páginas voy a tener un arreglo global de punteros a directorio, uno por cada tarea. Que va a ser inicializada durante el arranque del sistema operativo, y guardara la dirección de cada directorio.

### Código ASM

```
extern cant_bytes_leidos_o_escritos
syscall_ret: dd 0

sys_memoriaUtilizada:
    pushad
    push eax
    call cant_bytes_leidos_o_escritos
    mov [syscall_ret], eax
    add esp, 4
    popad
    mov eax, [syscall_ret]
    iret
```

### Código C

```
#define ultima_dir_codigo 0x04009000
#define dir_tope_aex 0x07FFC000
#define CANT_TAREAS 10
pd_entry* directorios[CANT_TAREAS];

int cant_bytes_leidos_o_escritos(int tarea){
    //Tomo los primeros 5 nibbles de ambas direcciones para recorrer.
    uint32_t dir_base = ultima_dir_codigo >> 12;
    uint32_t dir_tope = dir_top_aex >> 12;
```

```
//La inicializo en este valor ya que si no encuentro una,
//la resta entre ambas me devuelve cero.
uint32_t ultima_dir_accesed_encontrada = ultima_dir_codigo;

for(uint32_t i = dir_base; i < dir_tope; i++){
    if(is_accesed(&directorios[tarea], i << 12)){
        ultima_dir_accesed_encontrada = i << 12;
    }
}

return ultima_dir_accesed_encontrada - ultima_dir_codigo;
}

//Función auxiliar que chequea que una dirección virtual fue escrita o leída
//(bit atributo accesed encendido)
bool is_accesed(uint32_t cr3, uint32_t dir_virtual){
    uint32_t directory_index = dir_virtual >> 22;
    uint32_t table_index = (dir_virtual >> 12) & 0x000003FF;

    pd_entry* page_directory = (pd_entry*) cr3;

    pt_entry* page_table = (pt_entry*) (page_directory[directory_index].page_table_base << 12);

    return page_table[table_index].accesed;
}
```

Entradas en la IDT del servicio memoriaUtilizada:

- memoriaUtilizada:
  - OFFSET: &sys\_memoriaUtilizada
  - SELECTOR: GDT\_CODE\_RING\_0 << 3
  - ATTR: 0xEE00 (DPL=3, P=1, D=1 (32bits))

Nota: Como índice de interrupción en IDT aunque no lo especifico podría ser cualquiera entre 32 y 255 (que no coincida con lo de las interrupciones anteriores)